Introducción a R

Eduardo de Jesús Cuéllar Chávez

2/27/2022

En este manual podrás encontrar un pequeño introductorio a R

Instalar R

Primero hacemos click aquí: Descargar R

1. click en `download CRAN` en la barra izquierda
2. Escoge un sitio de descarga
3. Escoge tu sistema operativo
4. Click en `base`
5. Elige `Download R 3.0.3` y escoge las respuestas por default en cada pregunta

Instalar RStudio

1. Primero hacemos click aquí: Descargar R
2. Click en `Download RStudio Desktop`
3. Click en `Recommended For Your System`
4. Descarga el archivo y córrelo, te guiarán durante la instalación

Instalar Git

Instrucciones aquí

Configurar Github

Instrucciones aquí

Clonar repositorio de la clase

1. Abrimos `RStudio`
2. Vamos a `File-New Project`
3. Elegimos `Version Control`
4. Elegimos `Git`
5. En `Repository URL` pegamos el siguiente link:

`https://github.com/BlueRedem1/Stochastic-Processes-Simulation.git`

6. Escogemos un nombre y carpeta para el repositorio
7. Para actualizar nos vamos a la pestaña que dice `Git`, y le damos en la flechita azul que apunta hacia abajp

Básicos de R

Operaciones básicas

- Suma: `+`
- Resta: `-`
- Multiplicación: `*`
- División: `/`
- Exponencial: `^`
- Módulo: `%%`

Ejemplos:

```
#Suma
1+1
```

```
## [1] 2
```

```
#Resta
2-1
```

```
## [1] 1
```

```
#Multiplicación
3*3
```

```
## [1] 9
```

```
#División
10/5
```

```
## [1] 2
```

```
#Módulo
6%%2
```

```
## [1] 0
```

Workspace

¿Cómo veo en qué directorio estoy trabajando?

```
#getwd()
```

¿Cómo eligo mi directorio en el cual trabajo?

Hay dos maneras:

Usando la interfaz de RStudio

Nos vamos a `Session -> Set Working Directory -> Choose directory`

Usando funciones

```
# setwd("Aquí va el path completo del directorio en el cual queremos trabajar")
#ej:
setwd("/Users/cesarjuliocuellarruiz/Desktop")
```

Variables

Asignación

Usamos `<-` o bien `=`

```
x<-4
x=4
```

Si es más de un elemento, usamos `c()`

```
x<-c(4,5,6)
```

En R trabajamos con vectores ### Tipos de variables

- Caracter (Character)
- Numérico (Numeric)
- Entero (Integer)
- Complejo (Complex)
- Lógico (Logical)

Con la función `typeof` podemos ver el tipo de variable. Ejemplo:

```
typeof(x)
```

```
## [1] "double"
```

```
Verdadero<-T
Verdadero<-TRUE
Falso<-F
Falso<-FALSE
typeof(Falso)
```

```
## [1] "logical"
```

Librerías

Instalarlas

Usamos la siguiente función:

```
#install.packages("Aquí va el nombre de la paquetería que quieran instalar")
#Ejemplo:
# install.packages("tidyr")
#Es una librería para organizar de mejor manera los datos
```

Cargarlas

Antes de usarlas debemos cargarlas con la siguiente función:

```
#library(Aquí va el nombre de la paquetería que quieran llamar)
#Ejemplo:
library(tidyr)
```

¿Qué paqueterías son comunes?

El famoso tidyverse

- `ggplot2`: Gráficas bonitas
- `dplyr`: Manipulación de datos
- `tidyr`: Ordenar y acomodar la información
- `readr`: Formas más rápidas de leer datos
- `tibble`: Una alternativa a los dataframes

- `purr`: Un paquete de apoyo a la programación funcional
- `forcats`: Ayuda a tarbajar con variables categóricas
- `stringr`: Para trabajar mejor con strigs

Estadística

- `actuar`: Funciones actuariales y distribuciones de colas pesadas
- `astsa`: Análisis estadístico aplicado de series de tiempo
- `broom`: Convierte objetos estadísticos en tibbles
- `fitdistrplus`: Ayuda a ajustar una distribución paramétrica a datos censurados o no censurados (Análisis de supervivencia)
- `flexsurv`: Modelos flexibles de supervivencia paramétrica y multi-estados
- `forecast`: Ayuda a hacer predicciones para series de tiempo y modelos lineales
- `MASS`: Tiene unas funciones interesantes de estadística (Se llama, de hecho, MASS debido a las siglas de: Modern Applied Statistics with S, recordando que `S` es el lenguaje que dio origen a `R`)
- `rugarch`: Para ajustar modelos GARCH
- `survival`: Para hacer análisis de supervivencia
- `survminer`: Para hacer gráficos de análisis de supervivencia usando `ggplot2`
- `TSA`: Análisis de series de tiempo
- `nortest`: Tests de normalidad
- `moments`: Para sacar los momentos y algunos tests
- `lmtest`: Probar los supuestos de modelos de regresión lineal
- `imputeTS`: Para imputar datos daltantes en series de tiempo
- `fGarch`: Para hacer modelos GARCH

Archivos

- `readxl`: Para leer más rápido archivos de Excel

Finanzas

- `tidyquant`: Para hacer análisis financiero cuantitativo
- `timeSeries`: Objetos de series de tiempo financieras
- `QRM`: Herramientas con conceptos de administración de riesgos financieros

Estéticos

- `manipulate`: Con esta hacemos gráficos interactivos
- `shiny`: Para hacer aplicaciones interactivas en `R`
- `ggfortify`: Herramientas de visualización de datos para análisis estadístico
- `lattice`: Con este pueden escoger paletas de colores
- `rmarkdown`: Para correr este tipo de archivos (Para generar documentos en PDF deben tener instalado Tex en su computadora)

Y seguramente me siguen faltando más.

Especificar de qué liberería queremos una función

Para especificar la librería usamos el nombre esta, seguida de dos puntos, dos veces, y el nombre y de la fuunción.

```
#ejemplo
#library(MASS)
#MASS::truehist() #Es para hacer histogramas
```

Cargar datos (csv)

Podemos leer un csv con la siguiente funcion:

```
# read.csv("Aquí va la dirección relativa del archivo
#respecto al path seteado en el directorio de trabajo")
```

Por ejemplo, si tengo un archivo `ejemplo1.csv` en una la misma carpeta que tengo en el working directorio, bastaría con escribir `read.csv("ejemplo1.csv")`, si se encuentra en una carpeta distinta, tendría que escribir el path completo.

Escribir datos (csv)

Podemos escribir un csv con la siguiente función:

```
#write.csv(Aquí va el nombre del objeto que queremos
#escribir, como un dataframe)
```

Obtener ayuda

Si quieres averiguar más sobre alguna función o paquete basta con hacer uso de los siguientes comandos/funciones

```
?rnorm
help("rnorm")
```

Operadores

Lógicos

- `&` es para la operación "esta condicón y esta otra condición"
- `|`es para la operación "esta condicón o esta otra condición"

Comparaciones

- `==` es para comparar si dos objetos son iguales
- `!=` es para comparar si dos objetos son diferentes

Subsetting

- `[n]` para vectores, nos devuelve la entrada `n`
- `[i:k]` para vectores, nos devuelve desde la entrada `i` hasta la `k`
- `[-n]` para vectores, nos quita la entrada `n`
- `[[n]]` para listas, nos devuelve el objeto en la entrada `n`
- `[n,m]` para matrices o dataframes, nos devuelve el renglón `n`, columna `m`
- `[n,]` para matrices o dataframes, nos devuelve el renglón `n`
- `[,m]` para matrices o dataframes, nos devuelve la columna `m`
- `["Nombre_De_Columna"]` para matrices o dataframes, nos devuelve la columna de nombre `Nombre_De_Columna`

También podemos pasar un vector de booleanos o una condición, ej:

```
x<-c(1,2,3)
y<-c(T,F,T)
x[x>=2]
```

```
## [1] 2 3
```

```
x[y]
```

```
## [1] 1 3
```

Estructuras de control

while

```
i=1
while(i<3){
  i= i+1
  print(i)
}
```

```
## [1] 2
## [1] 3
```

if

```
if(i==3){
  print("Hola")
}else if(i<3){
  print("Adios!")
}else{
  print("Nos vemos")
}
```

```
## [1] "Hola"
```

for

```
for(numero in 1:5){
  print(numero)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Estructuras de datos

Vectores

Solo guardan un tipo de dato

```
x<-c(1,2,3)
x[4]<-4
x
```

```
## [1] 1 2 3 4
```

¿Desde dónde contamos? En `R`, comenzamos a contar a partir del `1`, no del `0` como en otros lenguajes de programación.

Ejemplo:

```r
x[1]
```

```
## [1] 1
```

```r
# x[0] #Nos marcará error
```

Listas

Guardan varios tipos de datos

```r
x<-list(c(4,5,6,7,8,9),T,4.5,"hola")
x
```

```
## [[1]]
## [1] 4 5 6 7 8 9
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] 4.5
##
## [[4]]
## [1] "hola"
```

```r
x[1] #Nos muestra todo el contenido del primer elemento, no podemos trabajar bien con este tipo, tenemo
```

```
## [[1]]
## [1] 4 5 6 7 8 9
```

```r
x[1][1] #¿Ven? ¡Es Null 1!
```

```
## [[1]]
## [1] 4 5 6 7 8 9
```

```r
x[[1]] #Aquí nos muestra el objeto del primer
```

```
## [1] 4 5 6 7 8 9
```

```r
#elemento en su forma original, aquí ya podemos trabajar con dicho objeto
x[[1]][1] #¡Aquí sí pudimos!
```

```
## [1] 4
```

Matrices

```r
x<-c(1,2,3,4)
mi_matriz1<-matrix(x,nrow = 2,ncol=2,byrow=F);mi_matriz1
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```r
mi_matriz2<-matrix(x,nrow = 2,ncol=2,byrow=T);mi_matriz2
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
mi_matriz1 %*% mi_matriz2 #Multiplicación de matrices
```

Operaciones con matrices

```
##      [,1] [,2]
## [1,]   10   14
## [2,]   14   20
```

```
mi_matriz1 * mi_matriz2 #Multiplicación entrada por entrada
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    6   16
```

```
mi_matriz1 + mi_matriz2 #Suma entrada por entrada
```

```
##      [,1] [,2]
## [1,]    2    5
## [2,]    5    8
```

```
mi_matriz1/mi_matriz2 #División entrada por entrada
```

```
##           [,1] [,2]
## [1,] 1.0000000  1.5
## [2,] 0.6666667  1.0
```

```
mi_matriz1
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
mi_matriz1[1,2] # Primero renglón, después columna
```

```
## [1] 3
```

```
mi_matriz1[1,] #Todo el primer renglón
```

```
## [1] 1 3
```

```
mi_matriz1[,2] #Toda la primera columna
```

```
## [1] 3 4
```

```
x<-c(4, 5, 6, 7, 8, 9)
y<-c("Hola","Grupo","Bonito","Día","Cuídense","Mucho")
mi_df<-data.frame("Numeros"=x, "Saludos"=y)
mi_df
```

Dataframes

```
##   Numeros  Saludos
## 1       4     Hola
## 2       5    Grupo
## 3       6   Bonito
## 4       7      Día
## 5       8 Cuídense
## 6       9    Mucho
```

```r
mi_df[1,2]
```

```
## [1] "Hola"
```

```r
mi_df[1,]
```

```
##   Numeros Saludos
## 1       4    Hola
```

```r
mi_df[,2]
```

```
## [1] "Hola"     "Grupo"    "Bonito"   "Día"      "Cuídense" "Mucho"
```

```r
mi_df["Numeros"]
```

```
##   Numeros
## 1       4
## 2       5
## 3       6
## 4       7
## 5       8
## 6       9
```

```r
head(mi_df,3) #Los primeros 3 renglones
```

```
##   Numeros Saludos
## 1       4    Hola
## 2       5   Grupo
## 3       6  Bonito
```

```r
tail(mi_df,3) #Los últimos 3 renglones
```

```
##   Numeros  Saludos
## 4       7      Día
## 5       8 Cuídense
## 6       9    Mucho
```

Funciones

```r
mi_funcion<-function(arg1, arg2=5){
  return(arg1+arg2)
}
mi_funcion(1)
```

Creemos nuestra primer función

```
## [1] 6
```

Generales

- `mean()`: Calcula la media
- `quantile()`: Calcula el cuantil x% de un conjunto de datos
- `sd()`: Calcula la desviación estándar
- `var()`: Calcula la varianza
- `sum()`: Calcula la suma de todas las entradas de un vector
- `exp()`: Exponencial
- `sqrt()`: Raíz cuadrada
- `table()`: Genera tablas de frecuencias

- `list()`: Crea una lista
- `sample()`: Obetener una muestra aleatoria de un conjunto de datos
- `replicate()`: Repetir un proceso varias veces
- `set.seed()`: Escoger una semilla para generar cosas aleatorias
- `matrix()`: Crea una matriz
- `library()`: Carga una librería
- `getwd()`: Obtenemos el directorio en el que estamos trabajando
- `setwd()`: Escogemos el directorio en el que queremos trabajar
- `read.csv()`: Cargamos un `.csv`
- `write.csv()`: Escribimos un `.csv`
- `round()`: Redondear hasta cierto decimales
- `ceiling()`: Redondear hacia arriba
- `numeric()` : Convertir a numérico
- `typeof()`: Ver tipo de dato
- `floor()`: Redondear hacia abajo
- `sapply()`: Aplicar una función a una lista o vector
- `Vectorize()`: Vectorizar una función
- `max()`: Obtener el máximo de un conjunto de datos
- `min()`: Obtener el mínimo de un conjunto de datos
- `pmax()`: Obtener el máximo de un conjunto de datos entrada por entrada
- `pmin()`: Obtener el mínimo de un conjunto de datos entrada por entrada
- `data.frame()`: Creamos un dataframe
- `install.packages()`: Instalamos una librería
- `help()`: Obtenemos información sobre una paquetería o función
- `log()`: Logarítmo natural
- `seq()`: Escribir una secuencia de números, de un número a otro con saltos de longitud definidos
- `sort()` Para ordenar un conjunto de datos

¡Y más!

De probabilidad

¿Qué podemos hacer?

- `r____(n)` Una muestra aleatoria de tamaño `n` de una disribución _____
- `d____(x)` La función de densidad de una distribución _____ evaluada en el punto `x`
- `q____(x)` El cuantil `x\%` de una distribución _____
- `p____(x)` La probabilidad acumulada hasta el punto `x`

En las líneas _____ debe de ir la distribución que queramos, escrita como se encuentran en el siguiente apartado

¿Qué distribuciones hay por default? **Importante**: ver la documentación usando `?` y una función para ver con detalle la información de los parámetros, ej: `?rbeta`

```
?rbeta
```

- Beta (**escribimos**`beta`): La escribimos como `beta` en `R`: Ej: `rbeta(n)` es una muestra de tamaño `n` de una distribución beta
- Binomial (**escribimos**`binom`): La escribimos como `binom` en `R`: Ej: `dbinom(x)` es la función de densidad de una binomial evaluada en el punto `x`
- Ji-Cuadrada (**escribimos**`chisq`): La escribimos como `chisq` en `R`: Ej: `pchisq(x)` es la probabilidad acumulada de una Ji-cuadrada el punto `x`
- Exponencial (**escribimos**`exp`): La escribimos como `exp` en `R`: Ej: `qexp(x)` es el cuantil x% de una exponencial
- Gama (**escribimos**`gamma`): La escribimos como `gamma` en `R` Ej: `pgamma(x)` es la probabilidad acumulada de una Gama el punto `x`

- Logística (**escribimos**`logis`): La escribimos como `logis` en `R`: Ej: `dlogis(x)` es la función de densidad de una logística evaluada en el punto `x`
- Nomal (**escribimos**`norm`): La escribimos como `norm` en `R`: Ej: `rnorm(n)` es una muestra de tamaño `n` de una distribución normal
- Poisson (**escribimos**`pois`): La escribimos como `pois` en `R`: Ej: `ppois(x)` es la probabilidad acumulada de una Poisson el punto `x`
- T de student (**escribimos**`t`): La escribimos como `t` en `R`: Ej: `qt(x)` es el cuantil x% de una T de student
- Uniforme *contínua* (**escribimos**`unif`): La escribimos como `logis` en `R`: Ej: `dunif(x)` es la función de densidad de una uniforme evaluada en el punto `x`
- Binomial negativa (**escribimos**`nbinom`): La escribimos como `nbinom` en `R`: Ej: `rnbinom(n)` es una muestra de tamaño `n` de una distribución binomial negativa
- Hyper geométrica (**escribimos**`hyper`): La escribimos como `hyper` en `R`: Ej: `rhyper(n)` es una muestra de tamaño `n` de una distribución hyper geométrica
- Multinomial (**escribimos**`multinom`): La escribimos como `multinom` en `R`: Ej: `rmultinom(n)` es una muestra de tamaño `n` de una distribución multinomial

¿Y la uniforme discreta? No viene implementada tal cual, pero podemos usar:

- Muestra aleatoria: `sample(i:k,n,replace=T)` para hacer una muestra de tamaño `n` de una uniforme con soporte en {i,i+1,...,k-1,k}
- Cuantil: `round(quantile(seq(i,k,by=1),x))` para obtener el cuantil x% de una uniforme con soporte en {i,i+1,...,k-1,k}
- Densidad: `ddu<-function(x,i,k) ifelse(x>=i & x<=k & round(x)==x,1/(k-i+1),0)`
- Acumulada: `pdu<-function(x,i,k) ifelse(x<i,0,ifelse(x<=k,floor(x)/(k-i+1),1))`

Recordemos que en la paquetería `actuar` podemos encontrar otras distribuciones que no vienen por default en `R`

Estadística

- `lm()` Para calcular un modelo de regresión lineal
- `glm()` Para calcular un modelo de regresión lineal generalizado
- `ks.test()` Para hacer la prueba Kolmogorov-Smirnov
- `chisq.test()` Para hacer el test de la Ji-cuadrada

Para graficar

- `plot()` Nos grafica una función o conjunto de datos

¡Listo!

Ahora tienes lo escencial de `R`, no olvides hacer tus cursos en Datacamp

Figure 1: ¡Bonito día!